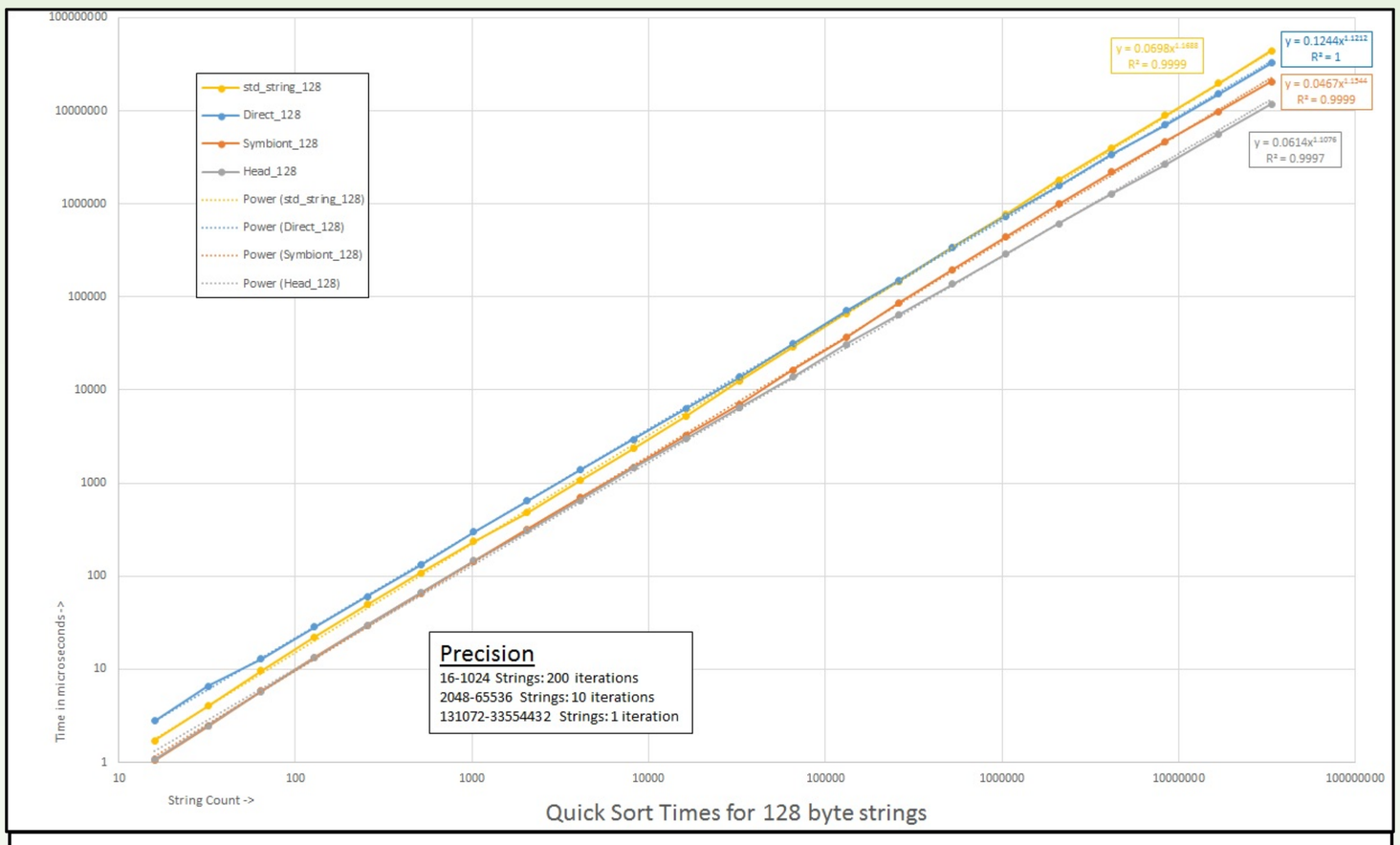
What We Need is a Green C++ Standard Library and a Green C++ Programming Model to Go with It.

C. Johnson, email: cjohnson@member.fsf.org

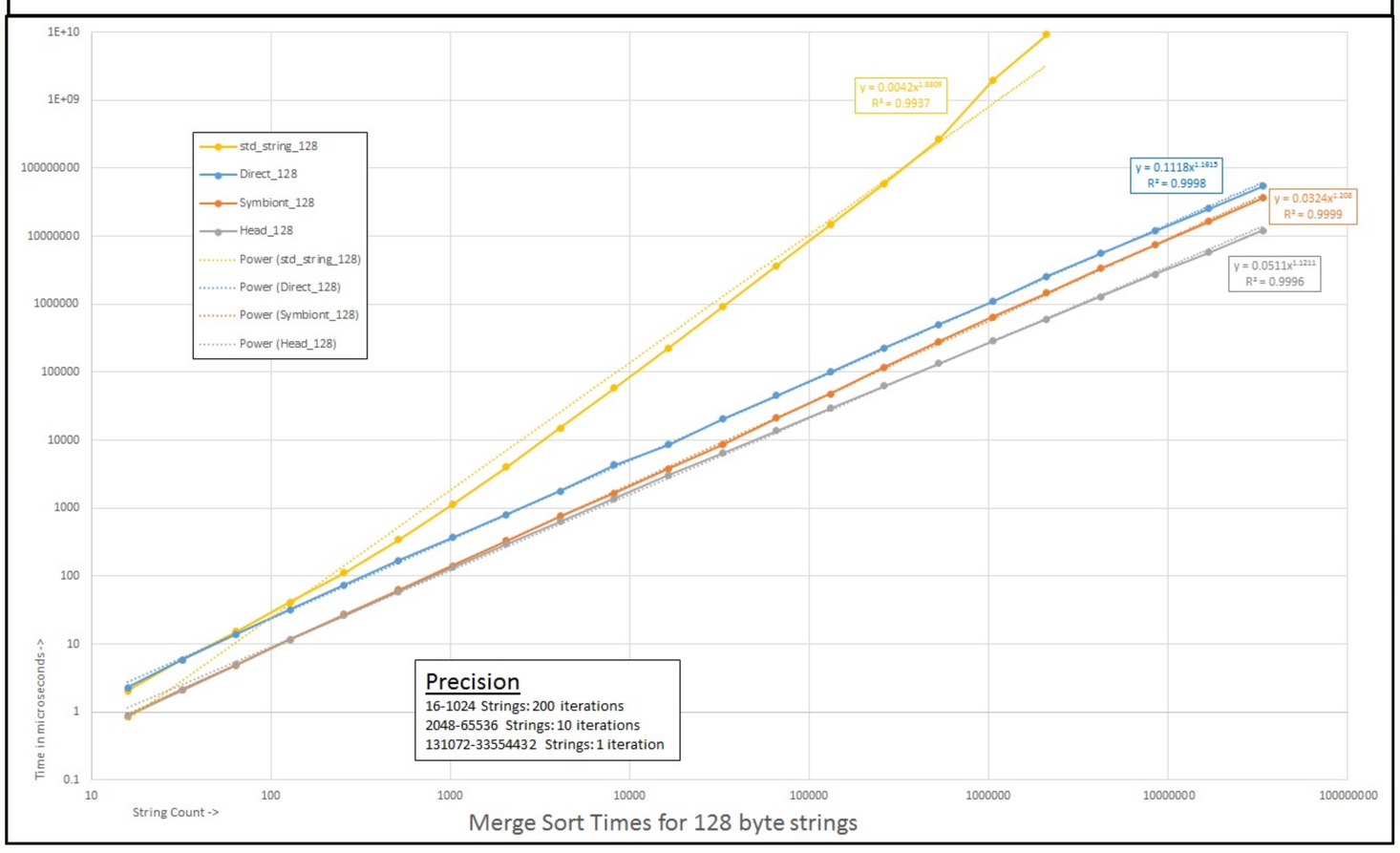
GitHub: https://github.com/charlesone/Kaldanes



C++ Standard strings quick sort almost an order of magnitude slower than one of the three new string classes at scale, and this is the best case, the worst case to the right

All execution numbers presented are single-threaded and built for release code, running on a small format HP (Intel 4-core i5-3570) box with 15GB of DRAM running Centos-7 Linux, with the gcc/g++ 4.8.5 C++11 compiler and runtime.

C++ Standard strings merge sort many orders of magnitude slower than all the new string classes at scale, since merge sort uses temporary arrays and std:strings have costly default initialization



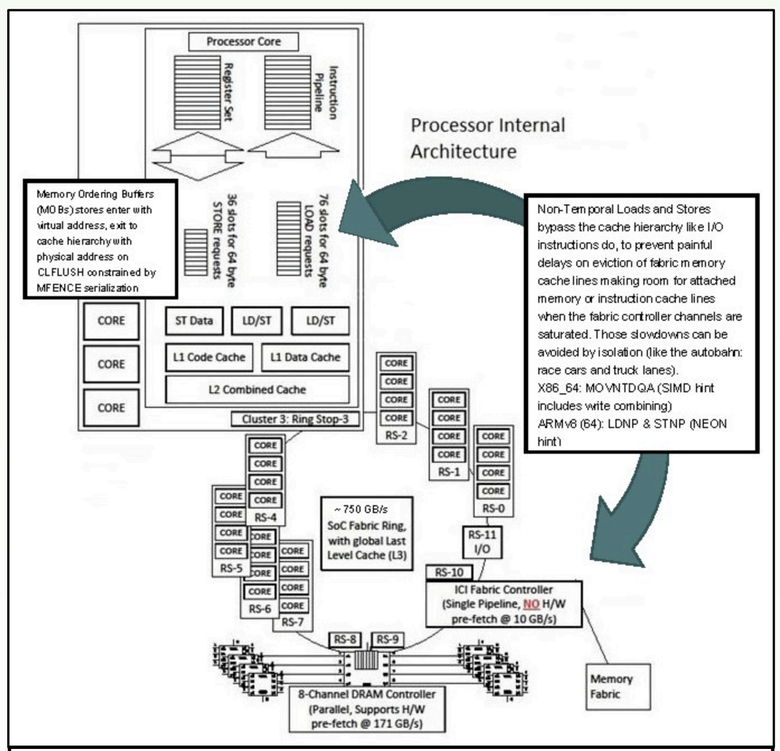
Thus, greener C++ overcomes the physical barriers to good C++ performance, to get C++ up to the level of typical C programming performance. This alone would require an entirely new C++ standard library to be written to stop wasting energy in the world's data centers, looming ominously to consume 20% of the world's power in the near future (see Pinar Tozun's HPTS 2015 talk.)

The second half of what's needed is a green C++ programming model dramatically surpassing the C program performance, based on variadic template parameter pack recursion, and generic programming in general, with C++11 features and something new that is dubbed "inference programming" or "inference computing," which executes in the compiler and not at runtime and involves compiling first the code and then the database into "memoized joined row objects" that are shrunken by a factor of 250 for 1000-byte row tables, using both type and data inference. What's left after C++ is done with compiling the code are heavily optimized pointer fetches and heavily optimized data movement, using "poor man's normalized keys" (Goetz Graefe) in the indexing.

The first half of what's needed is much faster and more importantly more scalable C++ classes and containers. Looking at the first two slides, we can start to graphically see what is wrong: replacing the complex C++ types and containers with simpler ones and dispensing with fine grained allocation, initialization and deallocation makes the code execute and scale remarkably well. Another boost comes from using a logical AND physical data format that is base + offset addressed to be allocated in slabs on the stack or mmapped in a DFS file system, Gen-Z librarian file, or 3D XPoint system, versus the slow execution and abominable scalability of standard library types and containers as they are serialized into and back out of messages, buffers and storage media. In a detailed discussion, some of the reasons why are still a standard library and C++ runtime mystery:

https://stackoverflow.com/questions/54562770/performance-of-big-slabs-due-to-allocation-initialization-and-deallocation-over

The fact is, that at the petabyte scale, it takes 302 days of computer time to allocate, default initialize and deallocate the standard library data structures, ***before adding any data to them***, versus 100 microseconds per 10 GB table (which comes to ten seconds per petabyte) using greener data structures, containers and programming models.



However, on a large Gen-Z memory-centric, 3D XPoint or mmapped DFS, it will be crucial to avoid polluting cache with large transfers